A Constraint-Based Planner for Data Production

Wanlin Pang[1] **Keith Golden**
NASA Ames Research Center
Moffett Field, CA 94035
{wpang, kgolden}@email.arc.nasa.gov

Abstract

This paper presents a graph-based backtracking algorithm designed to support constraint-based planning in data production domains. This algorithm performs backtracking at two nested levels: the *outer-backtracking* following the structure of the planning graph to select planner subgoals and actions to achieve them and the *inner-backtracking* inside a subproblem associated with a selected action to find action parameter values. We show this algorithm works well in a planner applied to automating data production in an ecological forecasting system. We also discuss how the idea of multi-level backtracking may improve efficiency of solving semi-structured constraint problems.

1 Introduction

Earth-science data processing (ESDP) at NASA is a data production problem of transforming low-level observations of the Earth system, such as remote sensing data, into high-level observations or predictions, such as crop failure or high fire risk. Given the large number of socially and economically important variables that can be derived from the data, the complexity of the data processing needed to derive them and the many terabytes of data that must be processed each day, there are great challenges and opportunities in processing the data in a timely manner, and a need for more effective automation. Our approach to providing this automation is to cast it as a planing problem: we represent data-processing operations as planner actions and desired data products as planner goals, and use a planner to generate data-flow programs that produce the requested data products.

Many of the recent advances in planning, such as state-based heuristic search or reduction to satisfiability problems, are not readily adapted to ESDP problems, due to some of its particular features, such as incomplete information, large and dynamic universes, complex data types, and complex constraints, just to name a few.

We take the approach, like many other researchers [van Beek & Chen, 1999; Lopez & Bacchus, 2003; Do & Kambhampati, 2001; Smith, Frank, & Jónsson, 2000], of translating the planning problem into a constraint satisfaction problem (CSP). However, since data processing domains are substantially different from other planning domains that have been explored, our approach to translating planning problems to CSPs differs as well. For example, [Do & Kambhampati, 2001] use variables to represent goals and domains to represent available planner actions achieving the goals. Constraints are used to encode mutual exclusion relations. While this is an effective approach for propositional planning problems, we also need variables to represent objects and action parameters, and constraints to represent relations among them. Thus, our encoding is somewhat more complex, and the CSPs resulting from our encoding are hard to solve by the search methods employed in other planners [van Beek & Chen, 1999; Lopez & Bacchus, 2003; Do & Kambhampati, 2001; Smith, Frank, & Jónsson, 2000].

We have developed a constraint-based planner, called DoPPLER, for **d**ata **p**rocessing **pl**anne**r**. From a data processing task, the planner constructs a *lifted planning graph,* from which it derives a CSP representation of the planning problem, and then searches the CSP for a solution. Whereas a conventional planning graph [Blum & Furst, 1997] is a grounded representation, consisting of ground actions and propositions, a *lifted* planning graph contains variables. This is not only a much more concise representation than an ordinary planning graph, but it also is the only practical way that we know to represent potentially infinite sets of ground actions. Even though the CSP derived from a lifted planning graph is difficult to solve by many existing CSP search methods such as chronological backtracking (BT), forward checking (FC) or conflict-directed backjumping (CBJ), it has certain structural properties inherited from the planning graph. We have developed a search algorithm based the structure of the planning graph to improve efficiency of solving the CSP.

In this paper, we report our work on applying DoPPLER to automating data production problem. We discuss how the data production problem is cast as a

[1] QSS Group Inc

planning problem which, in turn, is translated into a CSP, and how the planning graph is used to improve backtracking in solving the CSP. Section 2 discusses data processing as a planning task and our planning approach. Section 3 describes a graph-based CSP search algorithm that outperforms the standard search on problems with certain structural properties. Section 4 describes the graph-based planning search algorithm that is the main contribution of this paper. And Section 5 discusses related and future work.

2 Planning for Data Processing

Data processing is a task of transforming data products into other data products. A common sequence of data processing step is: 1) gather data from multiple sources; 2) convert the data into a common representation; 3) combine the data and perform other transformations; 4) feed the data into science models and then run the models; 5) convert the output of the model into some form suitable for visualization; 6) repeat some or all these steps depending on the requirements. To formalize data processing as a planning problem, we represent data-processing operations as planner actions, desired data products as planner goals, and available data sources as part of the initial state.

Planning in DoPPLER is a two-stage process. The first stage consists of a Graphplan-style reachability analysis [Blum & Furst, 1997] to derive heuristic distance estimates for the second stage, a constraint-based search. These stages are not entirely separate, however; constraint propagation occurs in both graph-construction and constraint search stages, and the graph is refined during the constraint-search phase.

2.1 Lifted Planning Graphs

From the planning problem specification, the planner incrementally constructs a directed graph, similar to a planning graph [Blum & Furst, 1997], but using a lifted representation (*i.e.*, containing variables). Arcs in the graph are analogous to causal links [Penberthy & Weld, 1992]. A causal link is triple $\langle \alpha_s, p, \alpha_p \rangle$, recording the decision to use action α_s to support precondition p of action α_p. However, instead of an arc to record a commitment of support, we use it to indicate the *possibility* that α_s supports p. The lifted graph contains multiple ways of supporting p; the choice of the actual supporter is left to constraint search. We add an extra term to the arc for bookkeeping purposes – the condition $\gamma_p^{\alpha_s}$ needed in order for α_s to achieve p. A link then becomes $\langle \alpha_s, \gamma_p^{\alpha_s}, p, \alpha_p \rangle$.

Given an unsupported precondition p of action α_p, our first task is to identify all the actions that could support p. Because the universe is large and dynamic, identifying all possible ground actions that could support p would be impractical, so instead we use a lifted representation, identifying all action *schemas* that could provide support. Given an action schema α, we determine whether it supports p by *regressing* p through α_s. The result of regression is the formula $\gamma_p^{\alpha_s}$. If $\gamma_p^{\alpha_s} = \bot$, then α_s does not support p. Initial graph construction terminates when all preconditions have support or (more likely) a potential loop is detected.

2.2 From Planning to Constraints

A constraint satisfaction problem (CSP) representing the search space is incrementally built during the planning graph construction. The CSP contains: 1) boolean variables for all arcs, nodes and conditions; 2) variables for all parameters, input and output variables and function values; 3) for every condition in the graph, a constraint specifying when that condition holds (for conditions supported by arcs, this is just the XOR of the arc variables); 4) for conjunctive and disjunctive expressions, the constraint is the respective conjunction or disjunction of the boolean variables corresponding to appropriate subexpressions; 5) for every arc in the graph, constraints specifying the conditions under which the supported fluents will be achieved (i.e., $\gamma_p^{\alpha} \Rightarrow p$, where γ_p^{α} is the precondition of α needed to achieve p) ; 6) user-specified constraints; and 7) constraints representing structured objects.

2.3 Planning Search

Guided by heuristic distance estimates extracted from the planning graph, the planner first selects planner subgoals to achieve and actions to achieve them, which form a *lifted plan*. After the subgoal and action selection, the the CSP solver finds values for variables representing planner action parameters. This is necessary to make actions executable. During the search, propagation is performed whenever a value is assigned to a variable. The search is an iterative process involving possible backtracks; that is, if there are no valid parameters for a chosen action, the planner has to search for another plan; if it is impossible to extract a plan from the current planning graph, the planning graph has to be extended, or the planner admits the failure of finding a plan.

2.4 A Simplified Example

A typical data processing task consists of gathering data files, transforming them, feeding them into a science model (e.g., a fire-risk model), and producing a final data file so that a decision maker can assess the fire risk of a particular region. For simplicity, we ignore much of the complexity of the data processing domain, and focus on one sub-problem: spatial aggregation. So a simplified task becomes to take some regions from thousands of available regions and compose them to create a mosaic that covers a specified region.

Specifically, a *region* is a pair of points (ul, lr) where ul is the upper-left corner and lr the lower-right corner. A point is a pair of coordinates (x, y). Normally x and y would be longitude and latitude, but as a further simplification, we will assume both x and y are non-negative integers. Further, we assume there are only 3 actions the planner may take, compose two regions horizontally (*comp2h*) and vertically (*comp2v*), and compose 4 regions (*comp4*) as in Figure 1.

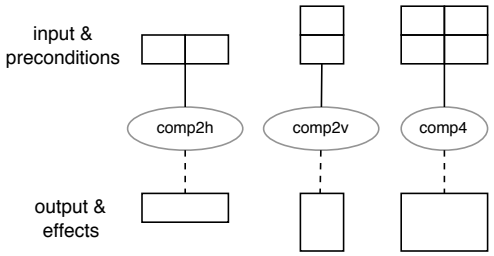

Figure 1: The planner actions

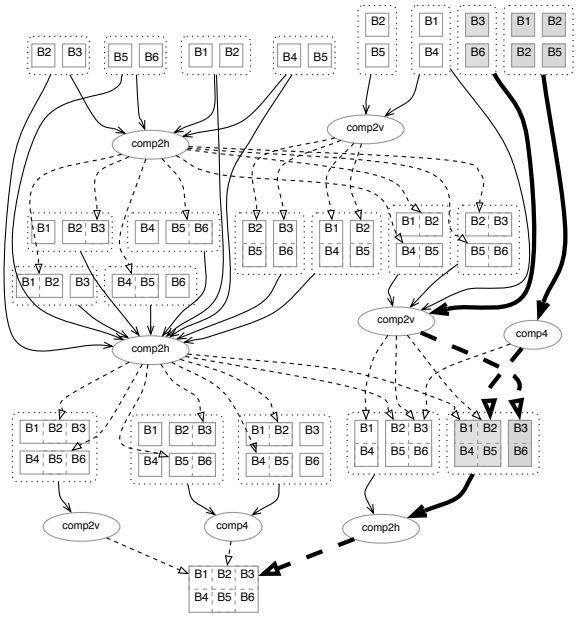

Figure 2: A compact search space: objects in a dotted rectangles are inputs to an action; an object divided by dashed lines is a composed object; single objects are available in the initial state.

A problem instance we consider here consists of some unit squares; that is, squares of (ul, lr) where $ul.x + 1 = lr.x$ and $ul.y + 1 = lr.y$. For example, $((0,0),(1,2)$, or $(2,3),(3,4))$. The goal is to compose a region covering $((0,0),(3,2))$.

The planning graph created by the planner is in Figure 2. At a high level, the planner finds a lifted plan by selecting subgoals and actions, shown in Figure 2 as a path from the initial state to the goal with dark arrows. This plan may not be executable because actions are not grounded. For example, the action *comp4* is selected because it has support from the initial state and it supports the action *comp2h*, but its parameters are not determined yet. The constraint solver then finds values for action parameters, which is shown in Figure 2 as groups of shaded rectangles.

It turns out that finding a lifted plan is a relatively easy task because it is a problem of finding a consistent assignment to a small number of variables in a very big constraint problem; whereas finding values for action parameters is a difficult CSP search problem. To address the issue, we developed a graph-based search algorithm, which is discussed in the rest of the paper.

3 Graph-Based Backtracking

A constraint satisfaction problem (CSP) consists of variables, domains that contain possible values the variables may take, and constraints that limit the values the variable can take simultaneously. In finite-domain CSPs (that is, every variable has a finite domain), a constraint over a variable subset can be represented extensionally as a subset of the Cartesian product of the domains of variables in the constraint. However, in the data processing domain we are interested in, the CSPs obtained from the planning problem contain variables that usually have infinite domains [Golden & Frank, 2002]. An infinite domain is represented as an an interval (for numeric types), regular expression (for string types) [Golden & Pang, 2003], or symbolic set (for object types). Because of infinite domains, the constraints are not represented extensionally as relations, but as procedures [Jónsson, 1996]. A procedural constraint consists of a set of variables (the scope) and a procedure (i.e., an `execute()` method) that can be executed to enforce the constraint by eliminating inconsistent values from the domains of variables in the scope. If execution of a constraint results in an empty variable domain, `execute()` returns failure, indicating the violation of the constraint.

Solving a CSP, in general, is NP-complete. However, many practical problems possess certain properties that allow tractable solutions. A class of structure-based CSP-solving algorithms, called decomposition algorithms, has been developed [Gottlob, 2000; Gyssens, Jeavons, & Cohen, 1994; Dechter, 1990; Dechter & Pearl, 1989]. Decomposition algorithms attempt to find solutions by decomposing a CSP into several simply connected sub-CSPs based on the underlying constraint graph and then solving them separately. Once a CSP is decomposed into a set of sub-CSPs, all solutions for each sub-CSP are found. Then a new CSP is formed where the original variable set in each sub-CSP is taken as a singleton variable. Usually the technique aims at decomposing a CSP into sub-CSPs such that the number of variables in the largest sub-CSP is minimal and the newly formed CSP has a tree-structured constraint graph. In this way, the time and space complexity of finding all solutions for each sub-CSP is bounded, and the newly formed CSP has backtrack-free solutions. The complexity of a decomposition algorithm is exponential in the size of the largest sub-CSP. The class of CSPs that can be decomposed into sub-CSPs such that their sizes are bounded by a fixed number k is tractable and can be solved by decomposition in polynomial time. This is the strength of CSP decomposition. A fatal weakness of CSP decomposition, however, is that the decomposition is not applicable to solving a CSP that is not decomposable, that is, its decomposition is itself. A secondary draw-

back of CSP decomposition is that, even if the CSP is decomposable, finding all solutions for all the sub-CSPs is unnecessary and inefficient.

Graph based backtracking (GBT) [Pang & Goodwin, 2003] was developed to address these issues. The idea of the GBT algorithm is to decompose the constraint graph into a tree of subgraphs (for example, *non-separable components*), and then search for a consistent assignment to variables involved in a subgraph and extend it to its children. At a subgraph where no consistent assignment can be found, GBT backtracks to the parent, reinstantiates variables in that subgraph, and starts from there. Within a subgraph, GBT searches for a consistent assignment to the variables in the subgraph in a way similar to standard backtracking, which may involve backtracks but limited to within the subgraph. The algorithm stops when a solution is found or when it proves that no solution exists. The detailed GBT algorithm can be found in [Pang & Goodwin, 2003]. In a simple way, the GBT algorithm performs backtracking at two nested levels: the *inner-backtracking* inside a subgraph and *outer-backtracking* following the subgraph tree obtained from the graph decomposition.

The GBT algorithm shares the merits of CSP decomposition and overcomes its weaknesses. As with the decomposition method, GBT decomposes the given CSP into sub-CSPs based on the underlying constraint graph decomposition. Unlike the decomposition method, however, GBT only tries to find one solution for a chosen sub-CSP, which is not separated from other sub-CSPs, and then tries to extend it to other sub-CSPs. If the underlying constraint graph can be decomposed, the complexity of GBT algorithm is bounded by the size of the largest sub-CSP; in the case that the constraint graph is not decomposable, GBT degenerates to a standard backtracking.

The GBT algorithm, as with other decomposition algorithms, depends on the underlying constraint graph representation and its decomposition. For details on graph representation and decomposition see [Gottlob, 2000]. For the planning problem at hand, we adapt GBT to utilize the planning graph structure for search heuristics.

4 Graph-Based Planning Search

Intuitively, the CSP derived from the planning graph is well structured due to the fact: i) the variables relevant to each planner action along with the subgoal it supports and the conditions enabling the action are tightly connected; ii) the constraints between variables of different actions are relatively sparse. Ideally, the CSP can be decomposed based on the constraint graph decomposition into sub-CSPs, each corresponding to a group of variables associated with a planner action. However, by experiments with a few graph decomposition methods, we haven't been able to decompose the constraint graph into a tree of subgraphs in a satisfactory way. It is still an on-going research effort to evaluate the process of translating the planning problem to a CSP aiming at optimizing the constraint problem in terms of its size and structural properties.

As an alternative, we decompose the CSP into sub-CSPs based on the planning graph instead of the constraint graph, each sub-CSP containing a group of variables that are relevant to a node in the planning graph representing a lifted action. In most of the cases, the sub-CSPs may not form a tree, which makes the traditional CSP decomposition methods inapplicable. However, the GBT algorithm can be adapted easily: the outer-backtracking is performed to select the planner subgoals and actions, the inner-backtracking to find values for action parameters by solving the associated sub-CSP. Even though the sub-CSPs do not form a tree, it is not a requirement for the graph based search approach but a preferred property.

4.1 Algorithms

The graph-based planning search algorithm is outlined in Algorithms 1 and 2. At a high level, the planner performs Best-First search to select the planner subgoals and actions achieving the subgoals. Once an action is chosen for a subgoal, it collects a subset of variables relevant to the action and calls a constraint solver SBT to find a consistent assignment for the collected variables. SBT performs a local backtracking to search for a solution to the sub-problem that is also consistent with solutions to the sub-problems preceding the current one. If SBT fails, the high-level search takes control, tries another action for the current selected subgoal or backtracks to a previously selected subgoal. At the end of selection of subgoals and actions, SBT is called again to find values for certain variables that may have been missed during the previous search.

Both algorithms interleave search with propagation, which is a process of continuously executing constraints as long as variable domains change. The propagation performs a partial *generalized arc-consistency* (GAC)[1] [Bessiere & Ch, 1997; Katsirelos & Bacchus, 2001], and it is an essential part of solving the constraint problem, not only because it reduces the search space by eliminating some inconsistent values, but also because the constraint problem at hand contains variables with infinite domains which cannot be enumerated by search. If executing a constraint fails, `propagate()` returns `failure`, which implies that the current value assignment to variables is inconsistent. Because the propagation is not limited to a sub-problem even if it is invoked by the SBT solving the sub-problem, it ensures the solutions to local sub-problems are globally consistent.

Comparing to the GBT algorithm in Section 3, the high-level BFS corresponds to the outer-backtracking; it backtracks when the selected best subgoal or action achieving a subgoal based on the planning heuristics is not feasible; that is, either the immediate propagation

[1] We call it partial GAC for two reasons: 1) not every constraint procedure enforces the GAC; and 2) not every constraint is executed in the propagation.

Algorithm 1 GBFS

Given a set of subgoals in the lifted planning graph G and a family of action sets $A = \{A(g)|g \in G\}$, each $A(g)$ is a set of actions achieving subgoal g. Let $G' \subseteq G$ be a set of active subgoals to be achieved (initially, the goals in the goal state), $P = (X, D, C)$ the CSP derived from the lifted planning graph, and X' a subset of searchable variables:

GBFS(G, A, P, G')

1. **while** $(G' \neq \emptyset)$ **do**
   (a) $g \leftarrow$ `a goal removed from` G'
   (b) **for** `each action` $a \in A(g)$
      i. **if** (`propagate`$(P, \{a\})$ `returns failure`) **continue** `for the next action`
      ii. $X' \leftarrow$ `variables relevant to` a
      iii. **while** (SBT(P, X') `returns success`) **do**
         A. $G_a \leftarrow$ `conditions of` a
         B. `add` G_a `to` G' `and sort` G'
         C. **if** (GBFS(G, A, P, G') `returns success`) **return** SBT(P, X)
      iv. **continue** `for the next action`
   (c) **return failure**
2. **return success**

Algorithm 2 SBT

Given a CSP $P = (X, D, C)$, and let $X' \subseteq X$ be a set of searchable variables:

SBT(P, X')

1. **if** $(X' = \emptyset)$ **return success**
2. `select` $x_i \in X$
3. **for** `each value` $v \in d(x_i)$
   (a) $x_i \leftarrow v$
   (b) **if** (`propagate(`P`,{`x_i`}) returns success`)
      i. `update` X'
      ii. **if** (`SBT(`P`,` X'`) returns success`) **return success**
4. **return failure**

fails, or the subsequent SBT search fails. The lower-level SBT is standard backtracking plus propagation for solving a specified sub-problem. Whereas the GBT algorithm and other graph-based decomposition methods require that the CSPs to be solved can be decomposed into tree-structured sub-problems, the multi-level backtracking algorithm presented here follows the structures of variable clusters, which may or may not form a tree, without an explicit decomposition. Such a multi-level backtracking strategy is particularly well-suited for semi-structured problems; though, more empirical studies are needed.

4.2 The Example Again

We take a look at the simplified example again and describe how the graph-based search algorithm works.

The task is to make a region covering $((0,0),(3,2))$, which consists of regions $B_1, B_2, ..., B_6$, all available from the initial state. At the beginning of the planning search, the active goal set G' contains only the top-level goal of making $((0,0),(3,2))$. Ignoring the search heuristics, we assume that the planner chooses the action *comp2h*, which composes two regions horizontally. These two regions are the parameters of the action, which are not determined initially. The inner-backtracking solver SBT is created with these parameters and it is called to find values for them. It quickly finds two regions, $((0,0),(2,2))$ and $((2,0),(3,2))$, for output parameters of action *comp2h*, and it also remembers its current status so that when it backtracks, it can find the next solution (the regions $((0,0),(1,2))$ and $((1,0),(3,2))$, see Figure 2) . The planner adds two subgoals, constructing the regions $((0,0),(2,2))$ and $((2,0),(3,2))$, to the active goals, and then continues recursively with the next best goal, which is one of the newly added two subgoals.

4.3 TOPS Application

We have applied the DoPPLER planner to the Terrestrial Observation and Prediction System (TOPS, http://ecocast.arc.nasa.gov) [Nemani *et al.*, 2002], an ecological forecasting system that assimilates data from Earth-orbiting satellites and ground weather stations to model and forecast conditions on the surface, such as soil moisture, vegetation growth and plant stress. The planner identifies the appropriate input files and sequences of operations needed to satisfy a data request, executes those operations on a remote TOPS server, and displays the results, quickly and reliably.

We have developed A TOPS planning domain, which specifies the data operations and data object types in TOPS. Data operations include running simulation-based models, reprojection, scaling, and construction of color composites, mosaics, and animations, etc. For object types in TOPS, Figure 3 shows the input and output objects to a TOPS model. To create a planning problem instance (i.e., a TOPS task), the user needs only to specify the planner goal, which is a description of a desired data product. A sample TOPS task would be something

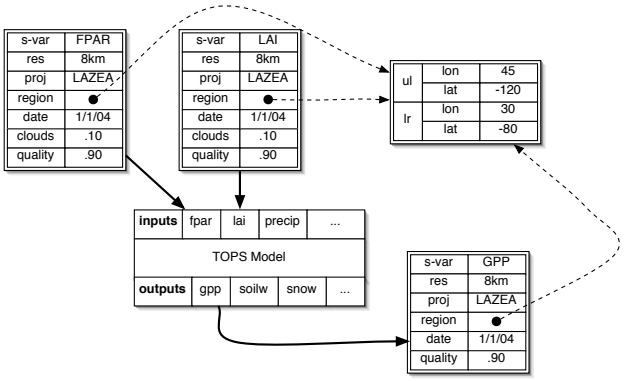

Figure 3: Structured inputs and outputs to a TOPS model

like "display Gross Primary Production (GPP) for continental US on May 5th, 2004".

The motivation of developing the graph based planning search is to speed up the search process so that the planner can produce the requested data product within a time limit acceptable to the user. Even though it is difficult for us to compare DoPPLER planner with publicly available planners, which cannot handle data-processing problem, we have compared DoPPLER to itself by turning on or off the inner-backtracking SBT. Without inner-backtracking SBT, for most of the TOPS tasks, it usually takes a few tries with different variable ordering heuristics to solve a problem; sometimes it fails within a specified time limit (e.g., 5 minutes). With inner-backtracking SBT turned on, the same TOPS tasks can be solved quickly without trying the additional variable ordering heuristics. However, we are currently conducting experiments on more TOPS tasks and artificial problems like the one in Section 2.4.

5 Conclusions

We have discussed the data production problem and how we reduce it to planning and solve the planning problem with a constraint search and propagation approach. A key element of our approach is the lifted planning graph, which we use as a basis for our CSP encoding, and use further to guide the planning and constraint search. The graph-based backtracking algorithm presented here has proved to be effective in our planner; it is also a general CSP solver that we intend to evaluate further on structured or semi-structured problems and to compare to other search and decomposition methods.

There has been little work in planner-based automation of data production. Two notable exceptions are Collage [Lansky, 1998] and MVP [Chien *et al.*, 1997]. Both of these planners were designed to provide assistance with data analysis tasks, in which a human was in the loop, directing the planner. In contrast, our planner does not require human interaction, which is appropriate for domains like TOPS, in which data production must be entirely automated; there is simply too much data for human interaction to be practical. Pegasus [Blythe *et al.*, 2003] is a workflow planning system for computation grids, a problem similar to ours, though their focus is on mapping pre-specified workflows onto a specific grid environment, whereas our focus is on generating the workflows.

References

[Bessiere & Ch, 1997] Bessiere, C., and Ch, J. 1997. Arc-consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI-97*, 398–404.

[Blum & Furst, 1997] Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *AIJ* 90(1–2):281–300.

[Blythe *et al.*, 2003] Blythe, J.; Deelman, E.; Gil, Y.; Kesselman, C.; Agarwal, A.; Mehta, G.; and Vahi, K. 2003. The role of planning in grid computing. In *Proc. 13th Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Chien *et al.*, 1997] Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1997. Using artificial intelligence planning to automate science data analysis for large image database. In *Proc. 1997 Conference on Knowledge Discovery and Data Mining*.

[Dechter & Pearl, 1989] Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

[Dechter, 1990] Dechter, R. 1990. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence* 41:273–312.

[Do & Kambhampati, 2001] Do, M., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence* 132:151–182.

[Golden & Frank, 2002] Golden, K., and Frank, J. 2002. Universal quantification in a constraint-based planner. In *AIPS02*.

[Golden & Pang, 2003] Golden, K., and Pang, W. 2003. Constraint reasoning over strings. In *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*.

[Gottlob, 2000] Gottlob, G. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence* 124:243–282.

[Gyssens, Jeavons, & Cohen, 1994] Gyssens, M.; Jeavons, P.; and Cohen, D. 1994. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence* 66:57–89.

[Jónsson, 1996] Jónsson, A. 1996. *Procedural Reasoning in Constraint Satisfaction*. Ph.D. Dissertation, Stanford University.

[Katsirelos & Bacchus, 2001] Katsirelos, G., and Bacchus, F. 2001. GAC on conjunctions of constraints. In *CP-2001*.

[Lansky, 1998] Lansky, A. 1998. Localized planning with action-based constraints. *Artificial Intelligence* 98(1–2):49–136.

[Lopez & Bacchus, 2003] Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In *Proceedings of IJCAI-2003*.

[Nemani *et al.*, 2002] Nemani, R.; Votava, P.; Roads, J.; White, M.; Thornton, P.; and Coughlan, J. 2002. Terrestrial observation and predition system: Integration of satellite and surface weather observations with ecosystem models. In *Proceedings of the 2002 International Geoscience and Remote Sensing Symposium (IGARSS)*.

[Pang & Goodwin, 2003] Pang, W., and Goodwin, S. D. 2003. A graph based backtracking algorithm for general CSPs. In *Proceedings of 6th Canadian Conference on Artificial Intelligence (CAI-2003)*, 114–128.

[Penberthy & Weld, 1992] Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. Principles of Knowledge Representation and Reasoning*, 103–114.

[Smith, Frank, & Jónsson, 2000] Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1):61–94.

[van Beek & Chen, 1999] van Beek, P., and Chen, X. 1999. CPlan: A constraint programming approach to planning. In *Proceedings of AAAI-99*.